Desarrollo de un programa de computador para el análisis lineal de estructuras aporticadas tridimensionales sometidas a cargas estáticas

Cristian Danilo Ramirez Vargas

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería Civil y Agrícola
Bogotá D. C., Colombia
2021

Desarrollo de un programa de computador para el análisis lineal de estructuras aporticadas tridimensionales sometidas a cargas estáticas

Cristian Danilo Ramirez Vargas

Tesis presentada como requisito parcial para optar al título de:
Magister en Estructuras

Director(a):
Ph. D. Martín Estrada Mejia

Línea de Investigación:
Análisis de estructuras
Grupo de Investigación:
Análisis, diseño y materiales - GIES

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería Civil y Agrícola
Bogotá D. C., Colombia
2021

Hombres inteligentes gran pensantes
Hemos creado un monstruo
Las bombas radiactivas y nucleares
Que descompondrán la humanidad
Quien totalmente se autodestruirá

Ya creador no hay para volver a comenzar
Como dijo la sagrada maldición
El universo en siete días lo creó

Razas de todos los colores
Tomemos una reacción
Potencias monopolizadoras
Analicen esta situación
Países tercermundistas
De brazos no nos crucemos
Acabemos pronto con esto

Futuro nunca habrá
Futuro nunca ha habido
Este en mundo que esta perdido
Dependiendo de un botón
Y de la decisión
De un idealista cabrón

La tercera guerra mundial
Será un estruendo nuclear
Donde historiadores no podrán narrarla
Y los humanos no podremos resistirla

Las invenciones científicas
Lejos de liberar de la ignorancia
Y del trabajo envilecedor
Lo aumentan
Y hacen más refinada la servidumbre

—*La ciencia de la autodestrucción*, La Pestilencia (1989)

Agradecimientos

No habría podido hacer este trabajo sin la dirección del profesor Martín Estrada. Su conocimiento del mundo de la programación me ayudó en momentos decisivos durante el desarrollo del código. Gracias a él trabajé con la librería *Three.js*. No sé como hacer para agradecerle por su paciencia.

Este trabajo también se debe al curso *Computación Visual* del profesor Jean Pierre Charalambos. Su descripción del *grafo* y como trabajar con la *escena* fue lo que me permitió hacer *FEM.js*.

Adicionalmente, apliqué el concepto de *cuaternión* en el problema de la rotación de los ejes de referencia tiempo después de haberlo estudiado en una de sus clases, lo que me permitió implementar el método de análisis matricial de manera innovadora. Gracias a su curso ahora creo entender muchas cosas que de adolescente siempre quise saber.

También quiero agradecer a la profesora Maritzabel Molina ya que mi entendimiento del método de análisis matricial proviene de su curso de *análisis estructural básico*. A ella nos debemos muchos ingenieros estructurales.

Así mismo, quiero agradecer al profesor Fernando Ramírez, de la Universidad de los Andes, por enseñarme el *método de los elementos finitos*, y al profesor Dorian Linero por enseñarme a implementarlo. A ellos gracias por haberme permitido ganar confianza con el método.

Finalmente, quiero agradecer la ayuda de la Coordinación Curricular del Posgrado en Estructuras, especialmente a la profesora Caori Takeuchi quien no tuvo reparos en dejarme ver el curso de Computación Visual. Ese día comenzó la verdadera tesis.

Contenido

Lista de Figuras

Lista de Algoritmos

1 Introducción

Los programas de computador comerciales para el análisis y diseño de estructuras que se encuentran vigentes a la fecha cuentan, en general, con un entorno gráfico que le permite al usuario describir el modelo de forma interactiva, procesarlo y visualizar los resultados de manera conveniente.

En Escamilla, 1995 se presenta una lista de algunos de estos programas de uso común en América Latina, entre los cuales se encuentra *ETABS* (Three Dimensional Analysis of Building Systems - Extended Version).

ETABS es un programa de computador creado por Edward Wilson, Jeffery Hollings y Henry Dovey en 1975. Según **ETABS1975**, este programa de computador fue desarrollado para el análisis estructural lineal de edificios de pórticos y muros a cortante sujetos tanto a cargas estáticas como sísmicas. El edificio es idealizado como un sistema de elementos aporticados y muros a cortante independientes interconectado por losas de entrepiso las cuales son rígidas en su propio plano.

Este programa es una extensión de *TABS* (Three Dimensional Analysis of Building Systems) para poder analizar pórticos tridimensionales. Según **ETABS1972**, una de las razones para desarrollar TABS fue darle una retroalimentación a los usuarios de los programas *FRMSTC* (Static Load Analysis of High-Rise Buildings), *FRMDYN* (Dynamic Analysis of Multistory Buildings), *LATERAL* y *SOLID SAP* (Static Analysis Program for Three-Dimensional Solid Structures).

FRMSTC permitía analizar edificios simétricos con pórticos y muros a cortante paralelos sujetos a cargas estáticas y evaluar los modos y las frecuencias. FRMDYN era similar a FRMSTC con la excepción que la carga era la aceleración del terreno debido a un desplazamiento dependiente del tiempo. LATERAL fue una extensión de FRMSTC que permitía analizar linealmente pórticos y muros a cortante que no eran necesariamente paralelos con tres grados de libertad en cada piso. SOLID SAP era un programa general de elementos finitos y tenía una opción que permitía introducir la aproximación de piso rígido. Este programa también tenía la opción de realizar análisis dinámico.

En la actualidad, ETABS se encuentra en la versión 18.1.1 y según **ETABS2020systemrequirements**,

puede ser ejecutado en computadores con sistema operativo Windows 7, Windows 8 o Windows 10 con arquitectura de 64 bits que cuenten como mínimo con un procesador Intel Pentium 4 o AMD Athlon 64, una resolución de 1024x768 pixeles con 16 bits por canal, 8 GB de RAM y 6 GB de espacio en el disco duro. En la figura **1-1** se presenta la ventana del programa ETABS ejecutándose en un computador con Windows 10.

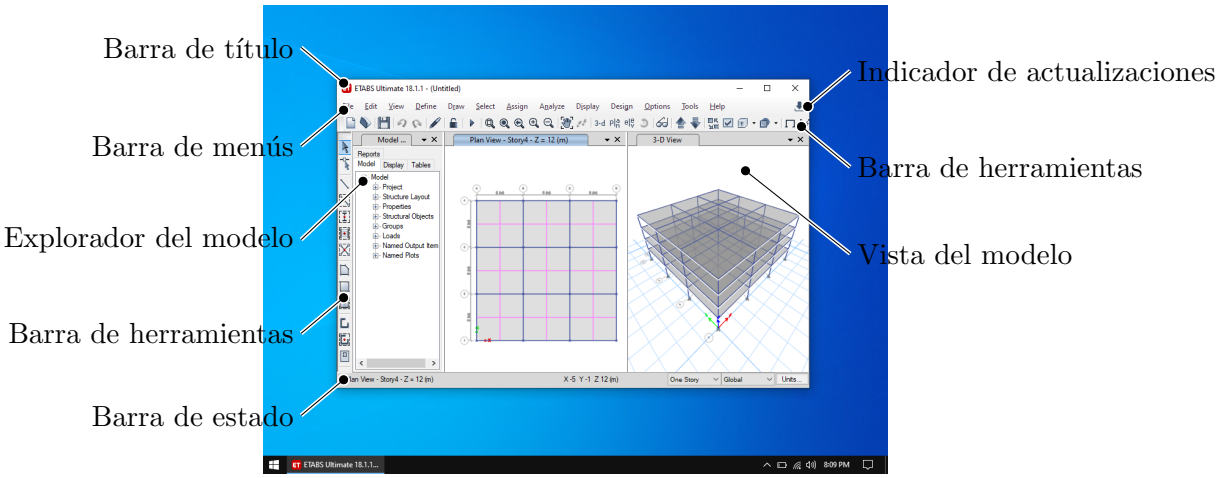

Figura 1-1: Ventana del programa ETABS ejecutandose en Windows 10.

A través de múltiples cuadros de diálogo, los cuales son accesibles ya sea a través de la barra de menús, las barras de herramientas, el explorador del modelo, las vistas del modelo o con atajos de teclado, el usuario es capaz de modelar la estructura que desea analizar al describir los materiales, las secciones transversales, los elementos estructurales, las condiciones de apoyo, los diafragmas y las cargas.

Según **ETABS2017analysisreferencemanual**, ETABS analiza el modelo usando el motor de análisis *SAPFire*, el cual es común a otros programas de la misma compañia (*SAP2000*, *SAFE* y *CSiBridge*). SAPFire es la última versión de la serie de programas *SAP* y ofrece las siguientes herramientas:

- Análisis estático y dinámico,
- Análisis lineal y no lineal,
- Análisis sísmico y análisis incremental no lineal (*pushover*),
- Análisis de cargas móviles,
- No linealidad geométrica, incluyendo efectos P-delta y grandes desplazamientos,
- Etapas constructivas,

- Fluencia lenta (*creep*), retracción (*shrinkage*) y envejecimiento,
- Análisis de pandeo,
- Análisis de densidad espectral de potencia y estado estacionario,
- Elementos aporticados y laminares, incluyendo el comportamiento de vigas, columnas, cerchas, membranas y placas,
- Elementos tipo cable y tendón,
- Elementos bidimensionales planos y elementos sólidos asimétricos,
- Elementos sólidos tridimensionales,
- Resortes no lineales y apoyos,
- Propiedades de los resortes y apoyos dependientes de la frecuencia,

Con los resultados del análisis del modelo, el posprocesador de ETABS puede *diseñar* los elementos estructurales de acuerdo a uno de varios códigos de diseño de diferentes países. ETABS es capaz de diseñar pórticos en acero, pórticos en concreto, vigas compuestas, columnas compuestas, vigas en acero de alma abierta (*steel joist*), muros a cortante y losas de concreto.

Adicionalmente, según **ETABS2019welcome**, ETABS cuenta con la posibilidad de generar dibujos estructurales esquemáticos de las plantas estructurales, de los despieces de vigas, columnas y muros a cortante, y de los detalles de las conexiones de acero.

En términos generales, estos programas de computador comerciales cuenta con características similares a las de ETABS. Actualmente, dichos programas están innovando para permitirle al usuario trabajar con modelos *BIM* (Building Information Modeling).

1.1. Objetivo

Desarrollar un programa de computador a código abierto para el análisis lineal de estructuras aporticadas tridimensionales sometidas a cargas estáticas.

Con este trabajo se pretende contribuir al ejercicio libre de la profesión del ingeniero estructural y a la enseñanza del análisis de las estructuras.

1.1.1. Objetivos específicos

- Desarrollar el módulo de análisis estructural para calcular el desplazamiento de los nudos, el valor de las reacciones y de las fuerzas internas de los elementos de una estructura sometida a cargas estáticas.

- Desarrollar el ambiente gráfico y la interfaz gráfica de usuario del programa de computador para permitirle al usuario ingresar los datos que describen la estructura, las acciones a las cuales se encuentra sometida y visualizar los resultados del análisis estructural.

1.2. Metodología

Se desarrollaron los programas de computador *pyFEM* y *FEM.js*. El primero para analizar estructuras aporticadas tridimensionales sometidas a cargas estáticas y el segundo para modelarlas. Esto con el fin que FEM.js pueda ser usado junto con otro programa de computador diferente a pyFEM.

Durante el desarrollo de estos programas, así como el de este documento, se utilizó *git* como sistema de control de versiones. Según **chacon2014git**, git es un sistema distribuido de control de versiones que registra los cambios realizados a un conjunto de archivos para coordinar el trabajo entre programadores.

Una copia de los repositorios de pyFEM y FEM.js se encuentran en la página de internet *GitHub*, la cual permite alojar proyectos utilizando git. pyFEM está alojado en https://github.com/rvcristiand/pyFEM mientras que FEM.js está alojado en https://github.com/rvcristiand/FEM.js.

1.2.1. pyFEM

pyFEM fue desarrollado en *Python*. Según Lutz, 2013, Python es un lenguaje de programación interpretado orientado a objetos cuya filosofía hace enfasis en la legibilidad de su código. Los archivos revelantes que componen el repositorio de pyFEM son:

```
pyFEM/
├── LICENSE
├── README.md
├── example_1.json
├── example_2.json
├── example_3.json
├── pyFEM/
│   ├── classtools.py
│   ├── core.py
│   └── primitives.py
└── test/
    ├── space_frame.py
    └── trusses.py
```

El archivo `LICENCE` contiene la licencia de pyFEM mientras que el archivo `README.md` contiene todas las instrucciones necesarias para ejecutar y usar pyFEM.

La carpeta *pyFEM* contiene los archivos `classtools.py`, `core.py` y `primitives.py`, los cuales contienen las instrucciones para analizar los modelos. La extensión *py* se usa para indicar que los archivos son programas de Python.

En el archivo `classtools.py` se encuentran las *clases* `UniqueInstances` y `AttrDisplay`, la primera para evitar que se creen *objetos* de una misma clase con la misma información y la segunda para generar una representación conveniente de los objetos.

En el archivo `primitives.py` se encuentran varias clases, entre ellas `Material`, `Section`, `Joint`, `Frame`, `Support`, `LoadPattern`, etc., las cuales permiten describir los diferentes atributos del modelo.

En el archivo `core.py` se encuentra la clase `Structure` la cual permite describir estructuras para ser analizados. Para crear objetos de esta clase se debe llamar la clase indicando los grados de libertad a tener en cuenta en el análisis. A partir de un objeto de esta clase es posible describir el modelo de la estructura al agregar materiales, secciones transversales, nodos, elemenetos aporticados, apoyos, patrones de carga, cargas en los nodos y cargas distribuidas en los elementos aporticados.

Los archivos `example_1.json`, `example_2.json` y `example_3.json` almacenan los modelos de tres de los ejemplos presentados en Escamilla, 1995 que han sido analizados con pyFEM. La extensión *json* se usa para indicar que los archivos tienen formato *JSON* (de sus siglas en inglés *JavaScript Object Notation*), el cual es un formato sencillo para el intercambio de datos. El modelo es descrito de tal manera que puede ser interpretado por FEM.js para

generar su representación en una escena tridimensional.

En el ejemplo 1.2.1 se presenta la solución a un ejercicio de Escamilla, 1995 usando pyFEM. En el capítulo 2 se presentan las rutinas que ejecuta pyFEM para solucionar los modelos estructurales.

Ejemplo

Resuelva completamente la cercha mostrada por el método matricial de los desplazamientos. El material es acero estructural con $E = 2040\ t/cm^2$. Las áreas están dadas entre paréntesis en cm^2.

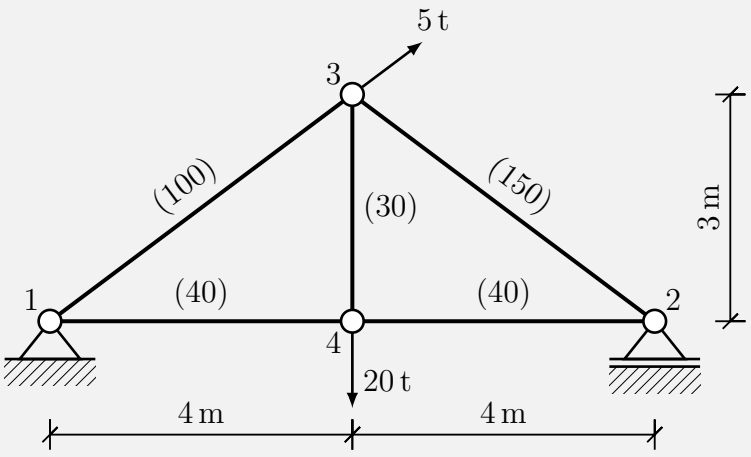

Figura 1-2: Cercha simple plana del *Ejemplo 7.1* de Escamilla, 1995.

Solución **-** En el algoritmo 1.1 se presenta un programa de Python para analizar el modelo de la estructura usando pyFEM. Las instrucciones consisten en crear un nuevo objeto tipo `Structure`, al cual se le ha dado el nombre *model*, agregarle (a) materiales, (b) secciones transversales, (c) nodos, (d) elementos aporticados, (e) apoyos, patrones de carga y (f) cargas en los nodos, analizar el modelo y exportarlo a formato JSON.

Cuando se ejecuta la instrucción `model.solve()` pyFEM comienza a solucionar el modelo de la estructura. Los pasos que efectúa para solucionar el modelo son: (1) asignar los grados de libertad de los nodos, (2) ensamblar la matriz de rigidez del modelo de la estructura, (3) imponer las condiciones de apoyo en la matriz de rigidez del modelo, (4) ensamblar el vector de fuerzas en los nodos para cada uno de los patrones de carga, (5) imponer las condiciones de apoyo en el vector de fuerzas en los nodos para cada caso de carga, (6) encontrar los desplazamientos de los nodos para cada patrón de carga, (7) encontrar las reacciones en los apoyos para cada patrón de carga y (8) guardar la solución en los nodos y en los apoyos para cada patrón de carga.

Algoritmo 1.1: Ingreso de los datos del modelo de la estructura a *pyFEM*.

```
# create the model
model = Structure(ux=True, uy=True)

# add materials
model.add_material(key='1', modulus_elasticity=2040e4)

# add sections
model.add_section(key='1', area=030e-4)
model.add_section('2', area=040e-4)
model.add_section('3', area=100e-4)
model.add_section('4', area=150e-4)

# add joints
model.add_joint(key=1, x=0, y=0)
model.add_joint(2, 8, 0)
model.add_joint(3, 4, 3)
model.add_joint(4, 4, 0)

# add frames
model.add_frame(key='1-3', key_joint_j=1, key_joint_k=3, key_material='1'
    , key_section='3')
model.add_frame('1-4', 1, 4, '1', '2')
model.add_frame('3-2', 3, 2, '1', '4')
model.add_frame('4-2', 4, 2, '1', '2')
model.add_frame('4-3', 4, 3, '1', '1')

# add supports
model.add_support(key_joint=1, ux=True, uy=True)
model.add_support(2, ux=False, uy=True)

# add load patterns
model.add_load_pattern(key='point loads')

# add point loads
model.add_load_at_joint(key_load_pattern='point loads', key_joint=3, fx=5
    * 0.8, fy=5 * 0.6)
model.add_load_at_joint('point loads' , 4, fy=-20)

# solve the problem
model.solve()
print(model)

# export the model
model.export('example_1.json')
```

Para realizar el ensamblaje de la matriz de rigidez del modelo de la estructura y del vector de fuerzas de los nodos, pyFEM asigna números a los grados de libertad de los nodos de la estructura en el orden en que fueron ingresados; al nodo 1 se le han asignado los grados de libertad *0* y *1*, al nodo 2 los grados de libertad *2* y *3*, y así sucesivamente.

Una vez se establecen los grados de libertad de los nodos se ensambla la matriz de rigidez del modelo de la estructura. Este proceso consiste en calcular una a una las matrices de rigidez de los elementos ensamblandolas en la matriz de rigidez del modelo.

El usuario puede consultar las matrices de rigidez de cada uno de los elementos del modelo de la estructura. En (1-1) se presenta la matriz de rigidez en coordenadas locales del elemento 1-3, la cual se obtiene mediante la instrucción `model.frames['1-3'].get_local_stiffness_matrix()`.

$$\begin{bmatrix} 40800 & 0 & -40800 & 0 \\ 0 & 0 & 0 & 0 \\ -40800 & 0 & 40800 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{t/m} \tag{1-1}$$

Así mismo, el usuario puede consultar las matrices de rotación de cada uno de los elementos del modelo. En (1-2) se presenta la matriz de rotación del elemento 1-3, la cual se obtiene mendiante la instrucción `model.frames['1-3'].get_rotation_matrix()`.

$$\begin{bmatrix} 0{,}8 & -0{,}6 & 0 & 0 \\ 0{,}6 & 0{,}8 & 0 & 0 \\ 0 & 0 & 0{,}8 & -0{,}6 \\ 0 & 0 & 0{,}6 & 0{,}8 \end{bmatrix} \tag{1-2}$$

El usuario tambien puede consultar las matrices de rigidez en coordenadas globales de cada uno de los elementos del modelo de la estructura. En (1-3) se presenta la matriz de rigidez en coordenadas globales del elemento 1-3, con sus respectivos grados de libertad, la cual se obtiene mediante la instrucción `model.frames['1-3'].get_global_stiffness_matrix()`.

$$\begin{array}{c} \\ 0 \\ 1 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 4 & 5 \end{array} \\ \begin{bmatrix} 26112 & 19584 & -261112 & -19584 \\ 19584 & 14688 & -19584 & -14688 \\ -26112 & -19584 & 26112 & 19584 \\ -19584 & -14688 & 19584 & 14688 \end{bmatrix} \end{array} \text{t/m} \tag{1-3}$$

En (1-4), (1-5) y (1-6) se presentan las matrices de rigidez en coordenadas globales de los elementos 1-4, 3-2 y 4-3 de la estructura las cuales se obtienen con instrucciones similares a la anterior.

$$\begin{array}{c} \\ 0 \\ 1 \\ 6 \\ 7 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 6 & 7 \end{array} \\ \begin{bmatrix} 20400 & 0 & -20400 & 0 \\ 0 & 0 & 0 & 0 \\ -20400 & 0 & 20400 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \text{t/m} \tag{1-4}$$

$$\begin{array}{c} \\ 4 \\ 5 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 4 & 5 & 2 & 3 \end{array} \\ \begin{bmatrix} 39168 & -29376 & -39168 & 29376 \\ -29376 & 22032 & 29376 & -22032 \\ -39168 & 29376 & 39168 & -29376 \\ 29376 & -22032 & -29376 & 22032 \end{bmatrix} \end{array} \text{t/m} \tag{1-5}$$

$$\begin{array}{c} \\ 6 \\ 7 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{cccc} 6 & 7 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 20400 & 0 & -20400 \\ 0 & 0 & 0 & 0 \\ 0 & -20400 & 0 & 20400 \end{bmatrix} \end{array} \text{t/m} \tag{1-6}$$

El usuario también puede consultar la matriz de rigidez del modelo de la estructura mediante la instrucción `structure.get_stiffness_matrix()`. En (1-7) se presenta la matriz de rigidez de la estructura.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 46512 & 19584 & 0 & 0 & -26112 & -19584 & -20400 & 0 \\ 19584 & 14688 & 0 & 0 & -19584 & -14688 & 0 & 0 \\ 0 & 0 & 59568 & -29376 & -39168 & 29376 & -20400 & 0 \\ 0 & 0 & -29376 & 22032 & 29376 & -22032 & 0 & 0 \\ -26112 & -19584 & -39168 & 29376 & 65280 & -9792 & 0 & 0 \\ -19584 & -14688 & 29376 & -22032 & -9792 & 57120 & 0 & -20400 \\ -20400 & 0 & -20400 & 0 & 0 & 0 & 40800 & 0 \\ 0 & 0 & 0 & 0 & 0 & -20400 & 0 & 20400 \end{bmatrix} \end{array} \text{t/m} \tag{1-7}$$

Una vez se ensambla la matriz de rigidez del modelo se modifica para tener en cuenta las condiciones de apoyo. Este proceso consiste en modificar las filas y las columnas

asociadas a los grados de libertad restringidos. En (1-8) se presenta la matriz de rigidez sujeta a las condiciones de apoyo del modelo de la estructura la cual se obtiene mediante la instrucción `model.get_stiffness_matrix_with_support()`.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 59568 & 0 & -39168 & 29376 & -20400 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -39168 & 0 & 65280 & -9792 & 0 & 0 \\ 0 & 0 & 29376 & 0 & -9792 & 57120 & 0 & -20400 \\ 0 & 0 & -20400 & 0 & 0 & 0 & 40800 & 0 \\ 0 & 0 & 0 & 0 & 0 & -20400 & 0 & 20400 \end{array}\right] \end{array} \text{t/m} \tag{1-8}$$

Una vez se obtiene la matriz de rigidez modificada del modelo de la estructura se resuelve para cada uno de los patrones de carga.

Así como se deben encontrar las matrices de rigidez de cada uno de los elementos del modelo de la estructura para posteriormente ensamblarlas, se debe encontrar la acción en los nodos de cada carga. En (1-9) se presenta el vector de fuerzas nodales del modelo para el patrón de carga *point loads* mediante la instrucción `structure.load_patterns['point loads'].get_f()`.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \\ \left\{\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 4 \\ 3 \\ 0 \\ -20 \end{array}\right\} \end{array} \text{t} \tag{1-9}$$

Obtenido el vector de fuerzas para dicho patrón de carga se imponen las condiciones de apoyo del modelo de la estructura. Debido a que los desplazamiento en los apoyos son iguales a cero el vector de fuerzas en los nodos no varia.

Al contar con la matriz de rigidez y el vector de fuerzas en los nodos, ambos modificados por las condiciones de apoyo, se calculan los desplazamientos y las reacciones. En (1-10)

y (1-11) se presentan el vector de desplazamientos y el vector de fuerzas en los nodos del modelo de la estructura para el patrón de carga *point loads*, los cuales son iguales a los presentados en Escamilla, 1995.

$$\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \begin{Bmatrix} 0 \\ 0 \\ 1{,}307 \\ 0 \\ 0{,}645 \\ -1{,}337 \\ 0{,}654 \\ -2{,}317 \end{Bmatrix} 1 \times 10^{-3}\,\mathrm{m} \tag{1-10}$$

$$\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \begin{Bmatrix} -4 \\ 7 \\ 0 \\ 10 \\ 4 \\ 3 \\ 0 \\ -20 \end{Bmatrix} \mathrm{t} \tag{1-11}$$

Cuando se ejecuta la instrucción `print(model)` pyFEM genera un informe del análisis. A continuación se presenta el informe generado para el objecto `model`.

```
Flag joint displacements
------------------------
ux: True
uy: True
uz: False
rx: False
ry: False
rz: False

Materials
---------
label   E                   G
1               20400000.0,     0

Sections
--------
label   A               Ix      Iy      Iz
1               0.003,  0,      0,      0
```

```
2               0.004,  0,      0,      0
3               0.01,   0,      0,      0
4               0.015,  0,      0,      0

Joints
------
label   x       y       z
1               0,      0,      0
2               8,      0,      0
3               4,      3,      0
4               4,      0,      0

Frames
------
label   Joint j Joint k material        section
1-3             1               3               1                       3
1-4             1               4               1                       2
3-2             3               2               1                       4
4-2             4               2               1                       2
4-3             4               3               1                       1

Supports
--------
label   ux              uy              uz              rx
    ry              rz
1               True,   True,   False,  False,  False,  False
2               False,  True,   False,  False,  False,  False

Load patterns
-------------
point loads:
label       fx          fy          fz          mx          my          mz
3             4.0         3.0           0           0           0           0
4               0         -20           0           0           0           0

Displacements
-------------
point loads:
label       ux          uy          uz          rx          ry          rz
1               +0.00000,       +0.00000,       +0.00000,       +0.00000,
          +0.00000,       +0.00000
2               +0.00131,       +0.00000,       +0.00000,       +0.00000,
          +0.00000,       +0.00000
3               +0.00065,       -0.00134,       +0.00000,       +0.00000,
          +0.00000,       +0.00000
4               +0.00065,       -0.00232,       +0.00000,       +0.00000,
          +0.00000,       +0.00000
```

```
Reactions
---------
point loads:
label          fx          fy          fz          mx          my          mz
1               -4.00000,        +7.00000,        +0.00000,        +0.00000,
          +0.00000,        +0.00000
2               +0.00000,        +10.00000,       +0.00000,        +0.00000,
          +0.00000,        +0.00000
```

Cuando se ejecuta la instrucción `model.export('example_1.json')` pyFEM genera un archivo que contiene toda la información del modelo en formato JSON para ser leído por FEM.js. A continuación se presenta la información del modelo en formato JSON.

```
{
    "materials": {
        "1": {
            "E": 20400000.0,
            "G": 0
        }
    },
    "sections": {
        "1": {
            "area": 0.003,
            "Ix": 0,
            "Iy": 0,
            "Iz": 0,
            "type": "Section"
        },
        "2": {
            "area": 0.004,
            "Ix": 0,
            "Iy": 0,
            "Iz": 0,
            "type": "Section"
        },
        "3": {
            "area": 0.01,
            "Ix": 0,
            "Iy": 0,
            "Iz": 0,
            "type": "Section"
        },
        "4": {
```

```
            "area": 0.015,
            "Ix": 0,
            "Iy": 0,
            "Iz": 0,
            "type": "Section"
        }
    },
    "joints": {
        "1": {
            "x": 0,
            "y": 0,
            "z": 0
        },
        "2": {
            "x": 8,
            "y": 0,
            "z": 0
        },
        "3": {
            "x": 4,
            "y": 3,
            "z": 0
        },
        "4": {
            "x": 4,
            "y": 0,
            "z": 0
        }
    },
    "frames": {
        "1-3": {
            "j": 1,
            "k": 3,
            "material": "1",
            "section": "3"
        },
        "1-4": {
            "j": 1,
            "k": 4,
            "material": "1",
            "section": "2"
        },
        "3-2": {
            "j": 3,
            "k": 2,
            "material": "1",
            "section": "4"
```

```
        },
        "4-2": {
            "j": 4,
            "k": 2,
            "material": "1",
            "section": "2"
        },
        "4-3": {
            "j": 4,
            "k": 3,
            "material": "1",
            "section": "1"
        }
    },
    "supports": {
        "1": {
            "ux": true,
            "uy": true,
            "uz": false,
            "rx": false,
            "ry": false,
            "rz": false
        },
        "2": {
            "ux": false,
            "uy": true,
            "uz": false,
            "rx": false,
            "ry": false,
            "rz": false
        }
    },
    "load_patterns": {
        "point loads": {
            "joints": {
                "3": [
                    {
                        "fx": 4.0,
                        "fy": 3.0,
                        "fz": 3.0,
                        "mx": 0,
                        "my": 0,
                        "mz": 0
                    }
                ],
                "4": [
                    {
```

```
                        "fx": 0,
                        "fy": -20,
                        "fz": -20,
                        "mx": 0,
                        "my": 0,
                        "mz": 0
                    }
                ]
            }
        }
    }
}
```

1.2.2. FEM.js

FEM.js fue desarrollado en *Three.js*. Según **dirksen2015threejs**, Three.js es un *API* (de sus siglas en inglés *application programming interface*) programada en *JavaScript* para *WebGL* que permite crear escenas tridimensionales en el navegador de internet. Los archivos revelantes que componen el repositorio de FEM.js son:

```
FEM.js/
├── LICENSE
├── README.md
├── css/
│   └── style.css
├── example_1.json
├── example_2.json
├── example_3.json
├── index.html
├── libs/
│   ├── CSS2DRenderer.js
│   ├── OrbitControls.js
│   ├── Projector.js
│   ├── dat.gui.min.js
│   ├── stats.js
│   └── three.js
├── main.js
└── modules/
    ├── FEM.js
    └── terminal.js
```

El archivo `LICENCE` contiene la licencia de FEM.js mientras que el archivo `README.md` con-

tiene todas las instrucciones necesarias para ejecutar y usar FEM.js.

El archivo `index.html` define la estructura de la página web de FEM.js. En la *etiqueta* `head` se define la ubicación los archivos `three.js`, `CCS2Renderer.js`, `OrbitControls.js`, `dat.gui.min.js`, `stats.js`, el estilo de la página según el archivo `style.css` y el *módulo* `main.js`. En la etiqueta `body` se definen las secciones `renderer-output` y `console`, para mostrar la escena tridimensional y recibir las instrucciones del usuario respectivamente (véase la figura **1-3**).

Los archivos `three.js`, `CCS2Renderer.js` y `OrbitControls.js` son necesarios para renderizar gráficos con WebGL, asociar objetos de la escena con etiquetas html y manipular la cámara. Estos archivos hacen parte del repositorio del proyecto Three.js alojado en GitHub (https://github.com/mrdoob/three.js/).

Los archivos `dat.gui.min.js` y `stats.js` permiten crear interfaces gráficas de usuario que cambian el valor de las variables y un monitor del desemepeño del código respectivamente. Estos archivos hacen parte de los repositorios *dat.gui* y *stats.js* alojados en GitHub (https://github.com/dataarts/dat.gui y https://github.com/mrdoob/stats.js/).

El archivo `style.css` define la presentación de las etiquetas html de la página web.

El archivo `main.js` define las funciones del archivo `FEM.js` que ejecuta `terminal.js` y algunos *eventos* para que todos los elementos de la página funcionen adecuadamente. El archivo `terminal.js` define una serie de funciones para interpretar y ejecutar las instrucciones que ingrese el usuario.

El archivo `FEM.js` contiene la configuración por defecto del programa, la descripción del panel lateral derecho y todas las funciones que hacen posible que el usuario pueda interactuar con el modelo.

Los archivos `example_1.json`, `example_2.json` y `example_3.json` almacenan los modelos de tres de los ejemplos presentados en Escamilla, 1995 que han sido generados con pyFEM.

FEM.js puede representar cualquiera de estos archivos al ejecutar la función `open()` con el nombre del archivo entre comillas dobles o sencillas como dato de entrada. En la figura **1-3** se presenta FEM.js con el archivo `example_2.json` abierto ejecutándose en el navegador de internet Firefox.

Así mismo, es capaz de ejecutar todas las funciones que se definan con `add_funcion()` del archivo `terminal.js`. Esta función recibe como parámetros el nombre de la función y un

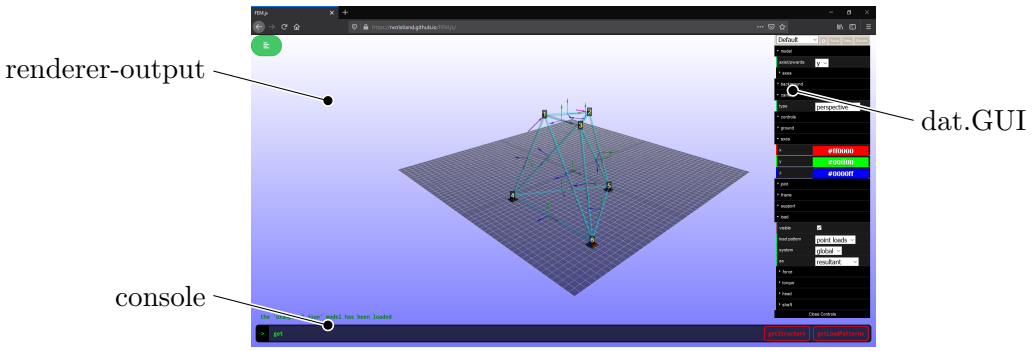

Figura 1-3: FEM.js ejecutándose en Firefox

objecto el cual debe definir la propiedad `func`. El nombre de la función se usa para llamar al parámetro `func` con los valores seprados por comas ingresados por el usuario.

A continuación se presentan una lista de las funciones definidas en `main.js`.

- addFrame,
- removeFrame,
- addSection,
- addRectangularSection,
- removeSection,
- addMaterial,
- removeMaterial,
- addJoint,
- removeJoint,
- setFrameView,
- showJointsLabel,
- hideJointsLabel,
- showFramesLabel,
- hideFramesLabel,
- setUpwardsAxis,
- setView,
- open,
- getStructure,
- getLoadPatterns.

Aunque el nombre de la función no tenga que ser necesariamente igual al del parámetro `func`, todos los nombres de la lista coínciden con funciones definidas en el archivo `FEM.js` (aun cuando no es necesario que estén definidas ahí).

La descripción de los parámetros de entrada de cada una de estás funciones se encuentran en el archivo `README.md`. A partir de dichas instrucciones es posible generar el modelo tridimensional de la estructura. Por ejemplo, para generar el modelo de la estructura del ejemplo 1.2.1 se deben ingresar las siguientes instrucciones

```
addMaterial(1, 2040e4)

addSection(1)
addSection(2)
addSection(3)
addSection(4)

addJoint(1, 0, 0, 0)
addJoint(2, 8, 0, 0)
addJoint(3, 4, 3, 0)
addJoint(4, 4, 0, 0)

addFrame(1-3, 1, 3, 1, 3)
addFrame(1-4, 1, 4, 1, 2)
addFrame(3-2, 3, 2, 1, 4)
addFrame(4-2, 4, 2, 1, 2)
addFrame(4-3, 4, 3, 1, 1)

addSupport(1, true, true)
addSupport(2, false, true)

addLoadPattern('point loads')

addLoadAtJoint('point loads', 3, 4, 3)
addLoadAtJoint('point loads', 4, 0, -20)
```

En la figura **1-4** se presenta FEM.js después de ejecutar los anteriores comandos.

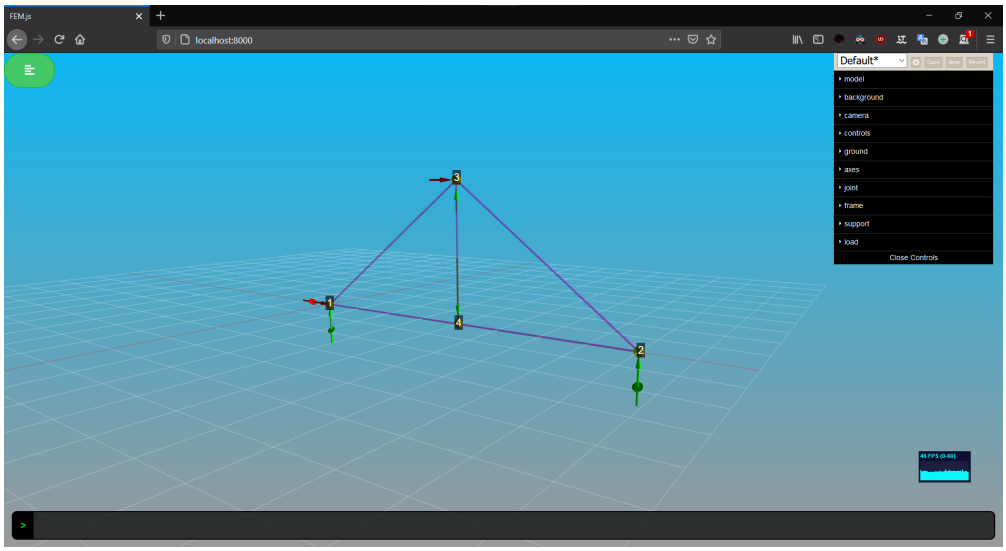

Figura 1-4: Ejemplo 1.2.1 modelado en FEM.js.

dat.gui

El panel lateral derecho de FEM.js fue desarrollado con *dat.GUI* para que el usuario pueda personalizar la escena. Este panel está agrupado en las siguientes categorias:

- model,
- background,
- camera,
- controls,
- ground,
- axes,
- joint,
- frame,
- support,
- load.

En la sección `model` se establece la orientación del modelo al definir uno de los ejes principales del modelo que apunta hacía la parte superior de la pantalla y una subsección llamada *axes*. En esta sección se define el tamaño y visibilidad de los ejes principales del modelo y dos subsecciones llamadas *head* y *shaft*. En estas susecciones se define la geometría de la cabeza y la cola de los vectores de los ejes principales del modelo.

En la sección `background` se establen dos colores para generar el fondo de la escena en gradiente. El color *top* define el color para la parte superior del fondo de la escena mientras que el color *bottom* define el color para la parte inferior.

En la sección `camera` se establece el tipo de proyección de la cámara pudiéndose elegir entre perspectiva y ortogonal. En la figura **1-5** se presenta el modelo del archivo `example_2.json` en proyección ortogonal.

En la sección `controls` se establece el comportamiento de los controles de FEM.js. Ahí se define la velocidad con la que estos hacen rotar, hacen *zoom*, desplazan la escena, si se desplaza la escena paralelo al plano del modelo o al plano de la proyección y una subsección llamada `damping`. En esta subsección se define si se adiciona un *amortiguamiento* a la rotación y la intensidad del mismo.

En la sección `ground` se define la visibilidad y el tamaño del conjunto de elementos *plano* y *grilla* así como dos secciones llamadas `plane` y `grid`. En la sección `plane` se define la visibilidad, el color, la transparencia y la opacidad del plano del modelo mientras que en la sección `grid` se define la visibilidad, el número de divisiones y los colores de las divisiones mayores y menores de la grilla.

En la sección `axes` se definen tres colores los cuales se asocian a los ejes x, y y z. Estos colores establecen los colores de los ejes globales y locales, los apoyos y las cargas. En la

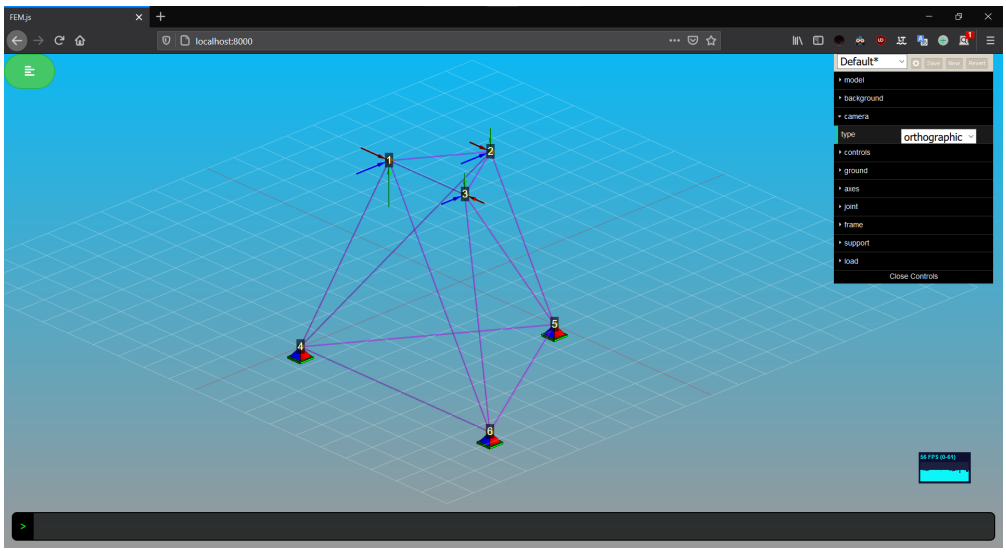

Figura 1-5: FEM.js en proyección ortogonal.

figura **1-6** se presenta el modelo del archivo `example_3.json` con una definición alternativa de dichos colores.

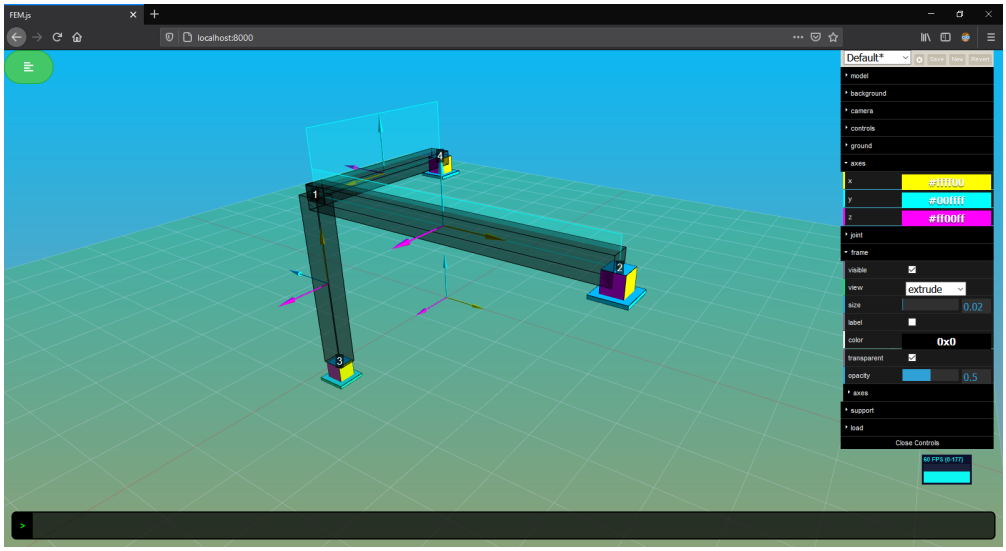

Figura 1-6: Colores alternativos para los elementos asociados a los ejes x, y y z.

En la sección `joint` se define la visibilidad, el tamaño, el color, la transparencia y la opacidad de los nodos del modelo. Así mismo se define la visibilidad de los *labels* de los nodos.

En la sección `frame` se define la visibilidad, la vista (*extruida* o en *palillo*), el tamaño, el color, la transparencia y la opacidad de los elementos aporticados del modelo. Así mismo se define la visibilidad de los *labels* de estos elementos y una sección llamada `axes`, similar a

la que se encuentra en la sección `model`, con la diferencia que esta establece la visibilidad y el tamaño de los ejes locales de los elementos aporticados. En la figura **1-7** se presenta el modelo del archivo `example_3.json` en *estructura de palillo*.

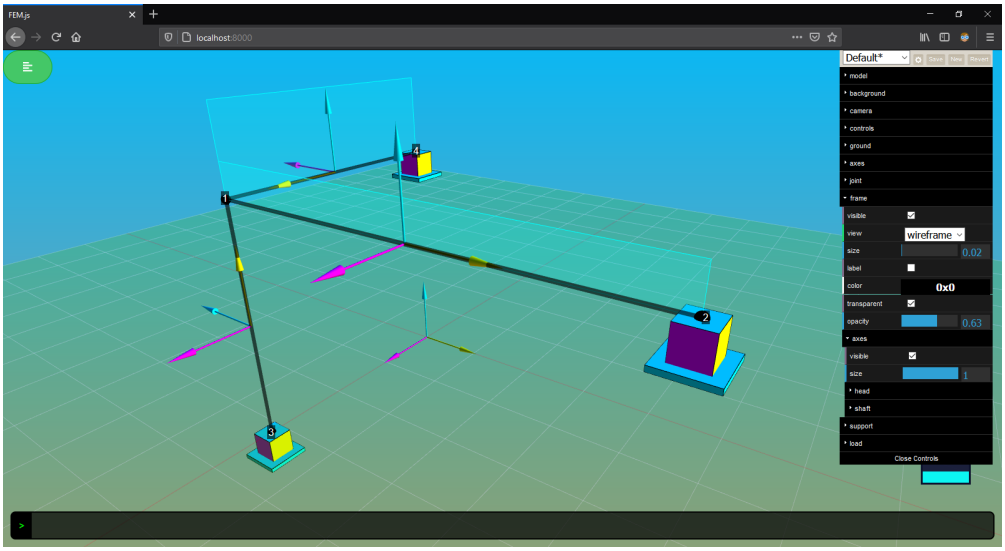

Figura 1-7: Vista del modelo como *estructura de palillo*.

En la sección `support` se define la visibilidad, el *modo* de los apoyos del modelo y dos secciones llamadas `analytical` y `space`. Los modos de los apoyos pueden ser *space* o *analytical*, en donde estos se representan con alguna de las analogías usadas en la literatura para representar apoyos o mediante vectores con un disco inclinado en la mitad de las colas.

En la sección `analytical` se definen tres secciones llamadas `head`, `shaft` y `restraint`, con las cuales se puede definir la geometría de los vectores con colas rectas o curvas (para representar restricciones a la traslación o a la rotación respectivamente) que tienen un un disco inclinado en la mitad de la cola.

En la sección `space` se definen tres secciones llamadas `foundation`, `pedestal` y `pin`, con las cuales se puede definir la geometría de los elementos *fundación*, *pedestal* o *rótula*, usados para representar los apoyos como elementos espaciales. Cuando se restringen todas las traslaciones y rotaciones el apoyo se resepresenta mediante un pedestal y una fundación, mientras que si se restrigen solo las translaciones el apoyo se representa por un pedestal y una pirámide con base cuadrada. Estos apoyos toman los colores definidos en la sección `axes` de manera conveniente.

En la figura **1-8** se presenta el modelo del archivo `example_3.json` con los apoyos en modo *analytical*.

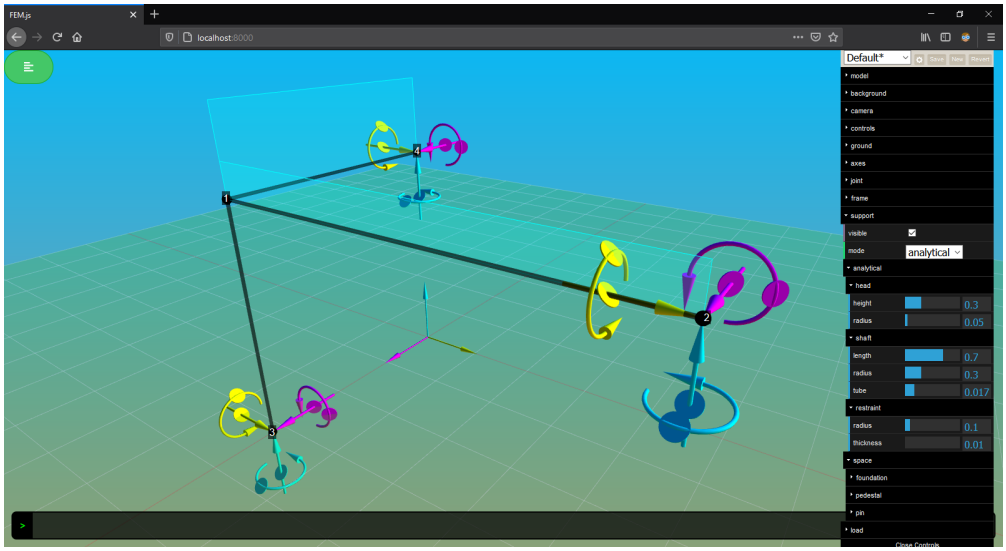

Figura 1-8: Apoyos del modelo en modo *analytical*.

En la sección `load` se define la visibilidad, el patrón de cargas, el sistema de coordenadas de referencia y como se representan las cargas del modelo, así como cuatro secciones con los nombres `force`, `torque`, `head` y `shaft`. En el momento únicamente se cuenta con el sistemda de refencia global para representar las cargas mientras que se pueden represetar como resultantes o componentes, aunque esta última opción actualmente sólo esta disponible para las cargas puntuales.

En las secciones `force`, `torque`, `head` y `shaft` se definen las dimensiones y el color de los elementos que representan las cargas. El tamaño de los diferentes elementos para representar las cargas se escalan en función de valor que estas representen.

2 pyFEM

En la figura **2-1** se presenta un elemento i tipo *pórtico* con sus nodos j y k empotrados. El sistema de coordenadas locales del elemento tiene como origen el nodo j. El eje x coincide con el eje centroidal del elemento y es positivo en el sentido del nodo j al nodo k. Los ejes y y z son los ejes principales del elemento de manera que los planos xy y zx son los planos principales de flexión. Se asume que el centro de cortante y el centroide del elemento coinciden de tal forma que la flexión y la torsión se presentan una independiente de la otra. Los grados de libertad se numeran del 1 al 12, empezando por las translaciones y las rotaciones del nodo j, tomados en orden x, y, z respectivamente.

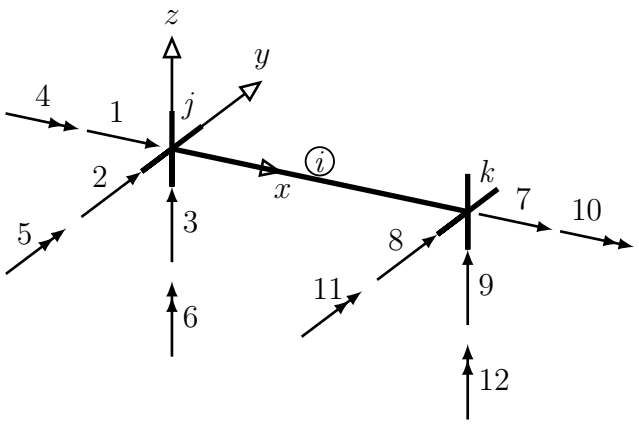

Figura 2-1: Elemento tipo pórtico en coordenadas locales.

Según Weaver y Gere, 1990, (2-1) es la matrix de rigidez del elemento tipo pórtico en coordenadas locales, donde E es el módulo de elasticidad del material, G es el módulo de elasticidad a cortante del material, L es la longitud del elemento y A_x, I_x, I_y y I_z son el área, la constante de torsión y los momentos principales de inercia con respecto a los ejes y y z de la sección transversal.

$$
\begin{array}{c}
\\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \\ \mathbf{10} \\ \mathbf{11} \\ \mathbf{12}
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccc}
\mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \mathbf{12}
\end{array} \\
\left[\begin{array}{cccccccccccc}
\frac{EA_x}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA_x}{L} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\
 & & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\
 & & & \frac{GI_x}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GI_x}{L} & 0 & 0 \\
 & & & & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\
 & & & & & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\
 & & & & & & \frac{EA_x}{L} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\
 & & & & & & & & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\
 & & & & & & & & & \frac{GI_x}{L} & 0 & 0 \\
 & & & & & & & & & & \frac{4EI_y}{L} & 0 \\
\textit{sim.} & & & & & & & & & & & \frac{4EI_z}{L}
\end{array}\right]
\end{array}
\tag{2-1}
$$

Según el *teorema de rotación de Euler* (véase Akademiia nauk SSSR., 1763), siempre es posible encontrar un diámetro de una esfera cuya posición es la misma después de rotarla alredor de su centro, por lo que cualquier secuencia de rotaciones de un sistema coordenado tridimensional es equivalente a una única rotación alrededor de un eje que pase por el origen.

El ángulo θ y el vector n que definen la rotación del eje x del sistema de coordenadas global hacia el eje x_m del sistema de coordenadas de un elemento se puede calcular como

$$
\begin{aligned}
&\mathbf{n} = (1,0,0) \times \mathbf{x_m} \\
&\theta = \operatorname{arc\,sen}((1,0,0) \cdot \mathbf{x_m})
\end{aligned}
\tag{2-2}
$$

Según Dunn, 2002, la rotación de un sistema de coordenadas tridimensionales alrededor del eje $\mathbf{n}$ una cantidad θ se puede describir mediante un *cuaternión* como

$$
\mathbf{q} = \begin{bmatrix} cos(\theta/2) & sin(\theta/2)\mathbf{n} \end{bmatrix} \tag{2-3}
$$

y se puede obtener la matriz de rotación a partir de un cuaternión de la siguiente manera

$$
\mathbf{R} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix} \tag{2-4}
$$

donde w es la parte escalar y x, y y z la parte vectorial del cuaternión.

Bibliografía

Akademiia nauk SSSR. (1763). *Novi comementarii Academiae scientiarum imperialis petropolitanae*. Typis Academiae Scientarum.

Dunn, F. (2002). *3D math primer for graphics and game development*. Plano, Tex, Wordware Pub.

Escamilla, J. (1995). *Microcomputadores en ingeniería estructural*. Santafé de Bogotá, ECOE Universidad Nacional de Colombia. Facultad de Ingeniera.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

Lutz, M. (2013). *Learning Python*. Sebastopol, CA, O'Reilly.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261-272. https://doi.org/10.1038/s41592-019-0686-2

Weaver, W. J. & Gere, J. (1990). *Matrix analysis of framed Structures*. New York, Van Nostrand Reinhold.